**Overall Plan**

* **Phase 1: HTML Structure.** We'll build the UI for the Traits card and its corresponding "Manage Traits" modal. This involves adding the list for current traits, input fields for new traits, and the modal structure itself. We will also update the version number.
* **Phase 2: Data Model & Core Logic (main.js).** We'll update the defaultNPC object to include a traits array, add a userDefinedTraits array to the bestiary metadata, and create the core functions for adding, updating, and reordering traits for an NPC. We will also write the logic for the "Manage Traits" modal.
* **Phase 3: UI Integration (ui.js).** We'll hook up the new HTML elements to our JavaScript. This includes creating functions to populate the trait lists, handling button clicks (add, delete, manage), and implementing the drag-and-drop sorting functionality.
* **Phase 4: Statblock Display (viewport.js).** Finally, we'll create the function to format the NPC's traits and add it to the statblock preview on the right.

**Phase 1: HTML File (npc engineer.html)**

This phase focuses on creating all the necessary user interface elements.

1. **Update Version Metadata:**
   * Find <meta name="version" content="0.05.05" />.
   * Change it to <meta name="version" content="0.06.00" />.
2. **Build the "Traits" Card UI:**
   * Locate the "Card: Traits" section.
   * Replace the placeholder <div> with a new structure containing:
     + A div with the ID npc-trait-list to display the list of the active NPC's traits. This is where the draggable items will go.
     + A dividing line (<hr>).
     + An input area with a datalist for the trait name (ID new-trait-name), allowing both selection from a list and free-text entry. The datalist itself will have the ID saved-trait-list.
     + A textarea for the trait description (ID new-trait-description).
     + An "Add Trait" button (ID add-trait-btn).
   * Add a "Manage Traits" button to the card header, similar to the one on the Languages card (ID manage-traits-btn).
3. **Create the "Manage Traits" Modal:**
   * Inside the main <div id="modal-overlay">, add a new div for the modal (ID manage-traits-modal).
   * This modal will be structured almost identically to the "Manage Languages" modal, containing:
     + A title and a close button.
     + An input for a new trait name (ID modal-trait-name).
     + A textarea for a new trait description (ID modal-trait-description).
     + An "Add" button (ID add-managed-trait-btn).
     + A div to list all saved, editable traits (ID managed-trait-list-div).

**Phase 2: Main Logic File (main.js)**

This phase handles the underlying data and logic.

1. **Update defaultNPC Object:**
   * Add a new property traits: [] to the baseDefaultNPC object.
   * In the healBestiary function, ensure traits is initialized as an empty array if it doesn't exist on an NPC.
2. **Update Bestiary Metadata:**
   * In the createNewBestiary function, add userDefinedTraits: [] to the metadata object.
   * In the healBestiary function, ensure metadata.userDefinedTraits is initialized if it's missing.
3. **Create Trait Management Functions:**
   * **addOrUpdateNpcTrait():** This function will take a name and description. It will check if a trait with that name already exists on the active NPC. If so, it updates the description; otherwise, it adds a new trait object ({name, description}) to the NPC's traits array.
   * **deleteNpcTrait(traitName):** Removes a trait from the active NPC's traits array by its name.
   * **reorderNpcTraits(oldIndex, newIndex):** A function to handle the drag-and-drop reordering of the traits array.
   * **addOrUpdateManagedTrait():** This will manage the bestiary-wide list of saved traits (userDefinedTraits). It will take a name and description from the modal, find if it exists, and either update it or add it.
   * **deleteManagedTrait(traitName):** Removes a trait from the userDefinedTraits array in the bestiary metadata.
4. **Display Logic:**
   * Create a new function calculateTraitsString(npc) that iterates through the npc.traits array and formats it into an HTML string for the viewport (e.g., <p><b><em>Trait Name.</em></b> Trait description.</p>).

**Phase 3: UI Interaction File (ui.js)**

This phase connects the HTML to the logic from main.js.

1. **Add New Element References:**
   * Add IDs for all the new elements from Phase 1 (the trait list, inputs, buttons, and modal elements) to the window.ui object.
2. **Implement UI Functions:**
   * **populateNpcTraitList():** This will clear the npc-trait-list div and repopulate it based on the activeNPC.traits array. Each item will be a draggable div containing the trait name, a delete button, and be clickable to populate the editing area.
   * **populateManagedTraitList():** This will populate the list inside the "Manage Traits" modal from activeBestiary.metadata.userDefinedTraits. Each item will have edit and delete buttons.
   * **populateTraitDatalist():** This will populate the <datalist> for the trait name input with the names from userDefinedTraits.
3. **Set Up Event Listeners:**
   * Create a new function setupTraitListeners().
   * Add a click listener to the add-trait-btn to call addOrUpdateNpcTrait() and then refresh the UI.
   * Add a click listener to the manage-traits-btn to show the modal.
   * Inside the modal, add listeners for its "Add" and "Delete" buttons to call the corresponding "managed trait" functions.
   * Add listeners to the NPC's trait list (npc-trait-list) for clicks on items (to populate the editor) and delete buttons.
   * Add a listener to the trait name input (new-trait-name) that, when an option from the datalist is selected, autofills the description box.
4. **Implement Drag-and-Drop:**
   * In populateNpcTraitList, set the draggable="true" attribute on each trait item.
   * Add dragstart, dragover, dragleave, and drop event listeners to the npc-trait-list container. These will handle tracking which item is being dragged and where it's dropped, finally calling reorderNpcTraits(oldIndex, newIndex) and refreshing the list. A temporary "drag-over" class can be used for visual feedback.

**Phase 4: Viewport Update (viewport.js)**

This is the final step to show the results.

1. **Integrate Traits into Viewport:**
   * In the updateViewport function, call the new window.app.calculateTraitsString(activeNPC) function to get the formatted HTML for the traits.
   * Add a placeholder in the main generatedHtml template string to inject the traits HTML, right before the "Actions" section would begin.

This phased approach ensures we build the feature logically, from the visual structure to the data handling, user interaction, and final display. Please let me know if you are happy with this plan.